2006

Free roaming: A system for ubiquitous computing

Huanjin Liu

FREE ROAMING:

A SYSTEM FOR UBIQUITOUS COMPUTING

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Huanjin Liu

March 2006

FREE ROAMING:

A SYSTEM FOR UBIQUITOUS COMPUTING

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Huanjin Liu

March 2006

Approved by:

[redacted]

Kay Zemoudeh, Ph.D. Chair, Computer Science Date 3/16/06

[redacted]

Tong Lai Yu, Ph.D.

[redacted]

Yasha Karant, Ph.D.

ABSTRACT

This document presented a free roaming system for a Ubiquitous Computing system. This Ubiquitous Computing system developed in CSUSB is intended to provide users with consistent user interfaces, and also release users from maintaining computer systems.

The Umbilical Cord system developed by J.E.Washawsky[1] performs as one cell of the Ubiquitous Computing system. It fulfills the purpose of Ubiquitous Computing system in one subnet. These cells can be deployed in different positions of a city, of a state, or even all over the world.

Based on the Umbilical Cord system, this project enriches the Ubiquitous Computing system with a Free Roaming system. This Free Roaming consists of a distributed authentication system, a data caching system and a communication system between them. It allows user to roam within this system and access his data everywhere. Together with the Umbilical Cord system, a Ubiquitous Computing system is functionally completed as a prototype, and is ready to be deployed into the Internet.

The distributed authentication system presented here is based on the RADIUS protocol[2]. It supports multi-master and multi-level authentication. These features increase the extensibility and availability of the system.

The distributed data caching system presented here is based on FTP protocol[3]. It supports file caching between multiple data servers in the Internet.

At the end, how to secure the system, how to backup the user data and how to improve system availability are discussed.

ACKNOWLEDGMENTS

I want to acknowledge the crucial assistance that my advisor, Dr. Kay Zemoudeh, gave to me during the process of completing this project. I could not have finished it without his encouragements.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION TO UBIQUITOUS COMPUTING

Purpose of this Document

This document describes the current state of the software system being developed in the department of Computer Science at California State University, San Bernardino. This system is based on the "Umbilical Cord" system, which was defined by J.E.Warshawsky[1]. Within a single subnet, the Umbilical Cord system allows a user to use any of the workstations. In this document, a "free roaming" system is presented. This free roaming system, which consists of a data caching sub-system and a roaming authentication sub-system, allows a user not only to roam within a subnet, but also to roam to different cities and even different states. Although many other topics need to be investigated and researched, with this system, a Ubiquitous Computing system is architecturally completed and ready to be deployed as a prototype.

Background

Ubiquitous Computing system is a different computing paradigm from either current popular personal computing or

the older mainframe computing. In the personal computing paradigm, every user typically buys his computer, installs the operating system and applications and uses the system. During the life time of this system, the user or his technical support needs to maintain it. The duty of system administration is placed on the user, including patching the operating system, installing and uninstalling various applications, etc. Usually the computing system the user has at home is different from the computing system he has at work. The user needs to learn and adapt to different computing systems while the basic functions in these systems are the same, like email, web browsing, and word processing.

In the mainframe computing paradigm, users use terminals to access the computing resources. In this case, all users of a mainframe system share one CPU (or CPU pool) and one memory system. Terminals do not have computing capability. Software applications are stored on disks attached to the mainframe computer, allowing multiple users to share the same applications while they share the same CPU. A drawback of this model is that if one user ran a CPU-intensive job, all other users would experience degraded performance. Another drawback of this paradigm is that a mainframe is too expensive compared to a Personal Computer.

On the other hand, Ubiquitous Computing system provides the user with a single interface and a unique set of applications, whether the user uses a computer at home or at work. At the same time, the burden of maintaining the operating system and applications is entirely transferred to very few computer engineers or administrators working remotely.

Specifically, Ubiquitous Computing system is a diskless client, remote server network-based system. The client is a diskless PC that can be deployed to places that are easy to be accessed by users, such as home, office, airport and hotel rooms. These clients are different from mainframe system terminals in that they have their own CPUs and large memory modules. All a client needs to do is to download the operating system and applications from a server when they are turned on. After a user is authenticated and authorized to use a client, his files are downloaded and he would use the client to just like a PC.

In this Ubiquitous Computing system, servers authenticate and authorize users to use the system, store operating system and applications, and store private data for users. Operating systems and applications will be maintained by administrators remotely. Users do not need to worry about upgrading and patching the operating system and

applications when new versions are released. They will always use the newest stable version of the operating system and applications maintained remotely by administrators. J.E.Warshawsky[1] compares this system with the satellite or cable TV system. Users just need to turn on the computing device and use them. Users do not need to upgrade their "TV set" and they do not know whether their "TV set" has been upgraded or not. These upgrades will be provided by the "cable" provider. All these devices have the same set of functions and the same set of interface wherever they are. Users will encounter the same look-and-feel and content wherever they go.

The basic concept of the Ubiquitous Computing System is illustrated in figure 1, which is contributed by J.E.Warshawshy[1]. In the diagram, two rectangles labeled with "Local Network" are omitted; each of them is a unit of "Umbilical Cord system". A user of this Ubiquitous Computing system can use the system in any of the "Local Networks" and can run applications he needs to access his data at any place.

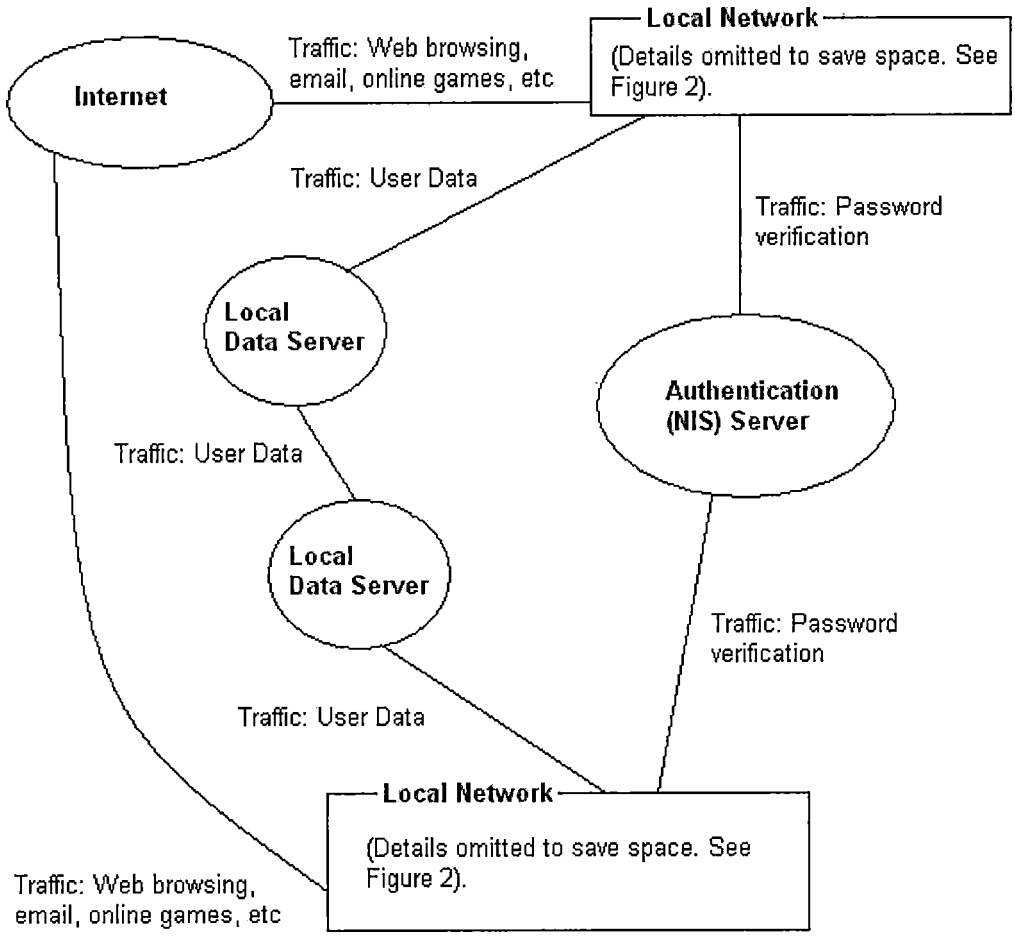

Figure 1. Topology of Ubiquitous Computing System

CHAPTER TWO

LITERATURE REVIEW

Based on Figure 1, to construct the Ubiquitous Computing system, we need many network elements. As mentioned before, J.E.Warshawsky[1] has implemented "Umbilical Cord" system, which is a unit of the Ubiquitous Computing system. Other than that, a "Free Roaming" system, which consists of a distributed authentication system and a data caching system, are needed for the Ubiquitous Computing system. In this chapter, works done by J.E.Warshawsky[1] and other researches related to the "Free Roaming" system will be discussed.

Umbilical Cord

In J.E.Warshawsky[1], the first step to Ubiquitous Computing system, a system named "Umbilical Cord" is implemented. The Umbilical Cord system is illustrated in Figure 2.

The Umbilical Cord system consists of clients, OS server and Data Server. These are described as follows:

1. Linux based diskless system

The client is a Linux based diskless system. When the user turns on the client, after BIOS initialization, the

PXE-aware Ethernet card tries to obtain an IP address using DHCP protocol. A PXE-configured DHCP server validates the Mac address of the incoming DHCP request and offers the IP address. At the same time, the DHCP server gives the IP address of a TFTP server. The client then will download the boot code from the server. After the download finishes the boot code takes over the control and downloads a configuration file from the TFTP server. This configuration file tells the client where to get the Linux kernel and basic root file system. The client downloads the Linux kernel and the root file system, and then boots up the Linux system.

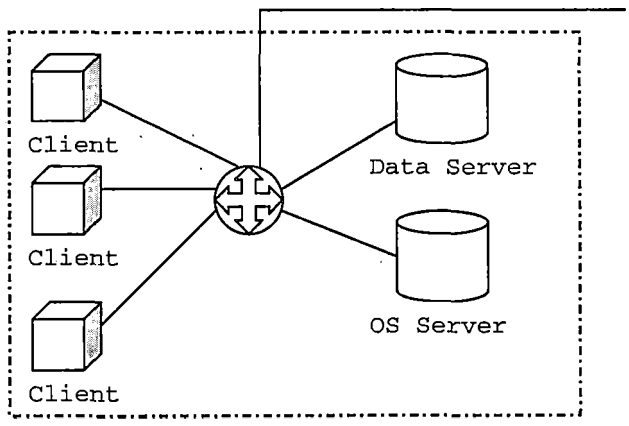

Figure 2. Umbilical Cord Topology

2. OS server and Data Server

In the Umbilical Cord system, the OS server hosts the Linux kernel and general applications to be downloaded by the client via the TFTP protocol. The OS server has a TFTP server running. The Data server hosts the users' home directories, which will be mounted using NFS file system after a corresponding user logs in successfully. The Data Server has an NFS server running on it.

3. Role of NFS and NIS

In the Umbilical Cord System, NIS is used to authenticate the user when he logs into the client. The root file system is mounted as a "stub" of the entire file system. It does not have everything for the client to run. The rest of the file system, including the users' home directories, will be mounted after the user logs in using NFS file system.

4. Scalable Swap

One of the most important contributions in the work of J.E.Warshawsky[1] is the scalable swap through NFS. Although we assume that the client has a large memory, there is always the chance that the client would run out of memory and need to perform paging and/or swapping. For a diskless system, the swapping usually is implemented using NFS. If each client served by the NFS server is configured to have a

fixed swap space, then all clients together will require a huge disk space from the server. Most often this requirement is prohibitive especially since this disk space is usually not used. For example, if a client has 1 GB of memory, 2GB of swapping space is usually assigned to it. 200 of these clients will need 400 GB of swap space on the server. Solution presented by J.E.Warshawsky[1] allows the server to allocate swap space for the client on demand. By providing a scalable swap space, a large amount of disk space, which is normally unused, can be saved.

Concept of Roaming

Roaming allows a user to be anywhere to use a device and/or service. For example, a mobile phone system enables a subscriber to travel to any place provided that there is a networked base station. In the Ubiquitous Computing system, "roaming" is not exactly the same as the mobile phone system because the user cannot move continuously unless a wireless network is used. But it does enable the user to travel from one subnet to another subnet, or even from one city to another city. To enable the roaming, at least two functions are needed in the Ubiquitous Computing system: a distributed authentication system and a distributed data caching system.

The distributed authentication system enables the user to login from any place. Since the Ubiquitous Computing

System is intended to be deployed over a country or even over the world, the ideal authentication system needs to have the following features:

1. Hierarchical Structure

Hierarchical structure is very straight forward and makes management easier for a large system. For a system that is intended to be deployed all over the world, a non-hierarchical system is impractical.

2. Multiple Master

In a master-slave structure, the database in the master changes dynamically when an entry of user information is added or deleted. Slaves replicate the database from the master periodically. The database on the slave does not change according to requests from clients. Because slaves are not full functional servers, a single master makes a system fragile.

3. Load Balancing and Disaster Recovery

For a system that is deployed for public use, supporting load balancing and disaster recovery are very important in case of multiple client requests at the same time and to recover from corrupted servers.

The distributed file caching system caches data for the user wherever he goes. Ideally, the user does not need to know where his data resides. All he knows is that wherever

he goes, he can access his data. That is to say, data caching is transparent to him.

The following sections review some existing authentication systems and distributed file systems, and investigate their features.

Current Distributed Authentication Systems

Kerberos Authentication

Kerberos[4] is a distributed authentication service that was originally developed at Massachusetts Institute of Technology to be used with the distributed computing system, Athena. In Kerberos, both client and server will prove to each other that they are really who they claim. Not all authentication systems discussed here have this feature. Most of them only authenticate the user for the server. The process of authentication is illustrated in Figure 3.

Light-Weight Directory Access Protocol

LDAP (Light-Weight Directory Access Protocol) was originally proposed in RFC 1777[5]. The latest version LDAPv3 "core" specifications includes RFCs 2251-2256 and 2829-2831. It is a client/server protocol that runs over TCP/IP stack and is used to store large amount of information in a hierarchical structure.

LDAP is often compared to a relational database. The difference between them is that the LDAP is designed to have

fast read but not fast write. So it is more appropriate for comparing information.

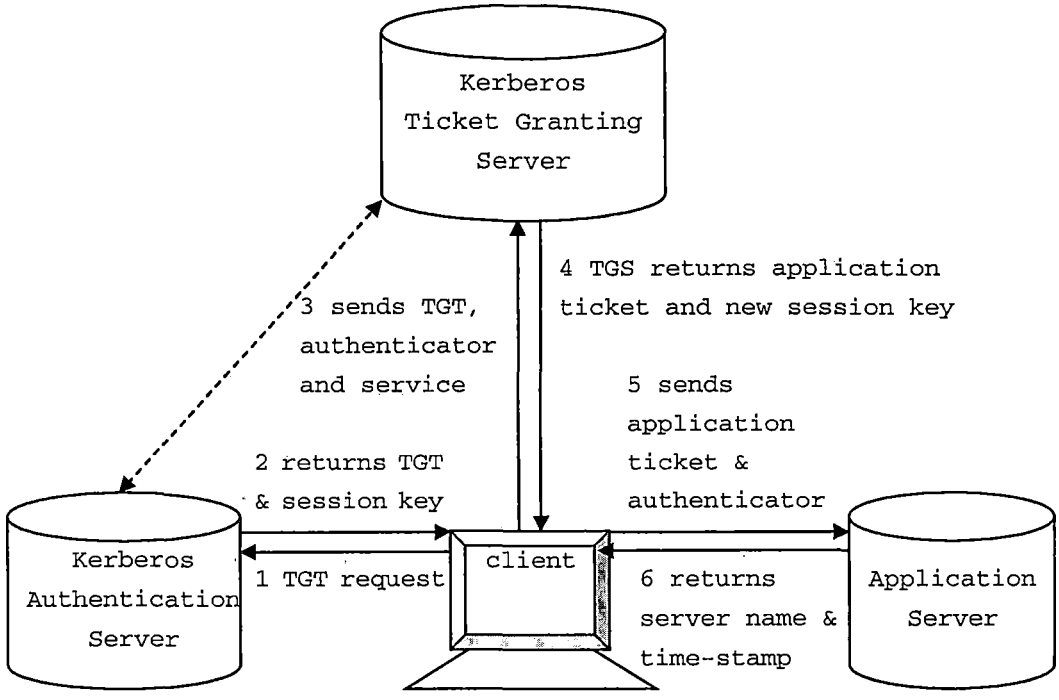

Figure 3. Kerberos Work Model

Active Directory

Active Directory[6] is a directory service provided by Microsoft's Windows 2000 Server family and its descendents. Active Directory runs on a Domain controller only. It is similar to and compliant with LDAP. In Active Directory, the network and its objects are organized and stored by

constructs such as domains, trees, forests, trust relationships, organization units and sites.

Active Directory is a service closest to the ideal authentication system that is discussed. It not only has a hierarchical structure, but also the ability to support multiple master servers. Active Directory supports multi-master replication between multiple domain controllers. By supporting multi-master, Active Directory is able to support load balancing and failure backup, which increases reliability and availability of the service. The only problem with this service is that it only runs on Windows servers.

Network Information System and Plus

Network Information System(NIS)[7] and NIS+ were introduced by Sun Mirosystems. Its original name was Yellow Pages but had to be changed because Yellow Pages is a register trademark of British Telecom. NIS is probably the most popular distributed authentication system in Unix/Linux systems.

NIS and NIS+ support a master-slave structure. The master hosts the user data and the slave replicates user data from the master. The slave can serve the authentication request from clients like the master, but it does not change the user information according to the incoming requests.

NIS has some shortcomings. First of all, only one master is allowed. Although slaves can process authentication requests from clients, they are not able to process requests to modify user information. Requests that change the management database have to be processed by the master. With the one-master structure, NIS is not an ideal authentication system for the Free Roaming system.

Remote Authentication Dial In User Service

Remote Authentication Dial In User Service (Radius) was described in RFC 2138[1] and renewed in RFC 2865[8]. It's a service intended for managing modem pools for dial-in users. Since a modem can be used to access the Internet, it has the requirements of security authentication, authorization and accounting (AAA). Radius is a service designed to fulfill these "AAA" requirements.

Radius uses a client/server model. The Network Access Server(NAS) acts as the client, and the Radius server works as the server. A Radius server can act as a client too if it is configured to work as a proxy server. Radius is very flexible and supports a variety of authentication methods. When a user name and password are provided, Radius server can authenticate the user by either PPP(Point-to-Point Protocol) PAP(Password Authentication Protocol) or CHAP(Challenge Handshake Authentication Protocol), or other

authentication methods. Radius is also an extensible protocol. It can query the user information either from an SQL database or LDAP.

Roaming of a Mobile Communication System

Mobility Management of Mobile Communication systems such as PCS (Personal Communication System), GSM (Global System for Mobile Communication) or CDMA (Code-Division Multiple Access), are based on IS-41 protocol from TIA (Telecommunications Industry Association)[9][10].

When a user subscribes to a cellular telephony system, this system becomes the user's home system. A subscriber's entry will be added to the system's database, which is the HLR (Home Location Register). When this user visits another cellular service area, a temporary entry will be created in that system's VLR (Visitor Location Register). This VLR is used to retrieve information of visiting users.

Suppose that a user's home system is at Los Angeles, so his HLR is in Los Angeles. When he visits New York, he must register either automatically or manually in the VLR of New York. The VLR sends messages to user's HLR to inform a successful registration. The HLR needs to change the new location of the user. When the user makes a phone call, the mobile phone will first communicate with the local MSC (Mobile Switching Center). Then the call will be forwarded

to the VLR. Finally, the call will be handled and forwarded to a PSTN (Public Switched Telephone Network) or another mobile phone. If a PSTN user or a mobile user calls the number, the switch will query the user's HLR for his current VLR. The HLR will then forward the call to the current VLR. IS-41 can use either share key or public key authentication system.

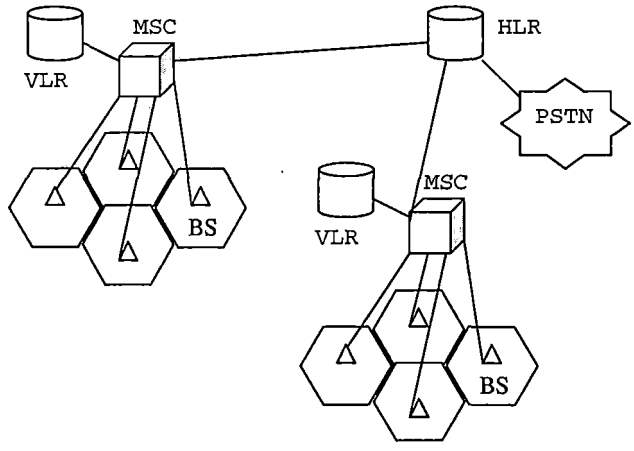

Figure 4. Architecture of Cellular Network

The Free Roaming system and mobile communication system have some common characteristics:

1. Allow users to roam inside the system;
2. Need to authentication users from different access points;

3. Need to check users' home database to get his information.

A major difference between them is that when a user roams in the Free Roaming system, his data needs to be moved from his home data server to the local data server. In a mobile phone system, this is not needed.

Authentication of Wireless Network

S.Keshi-Kasari et al[11] provided a valuable Radius-based model for the Free Roaming system. In their model, Radius proxy server is used to construct a hierarchical authentication system. As shown in figure 4.

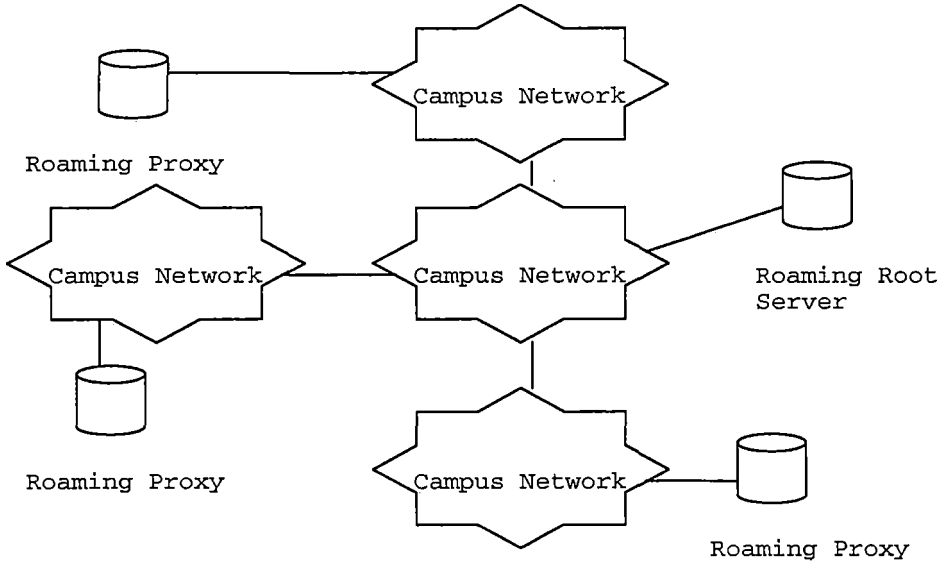

Figure 5. Roaming Hierarchy for Wireless Network

The original purpose of this structure is to support students and scholars traveling between these campuses either using wireless or wired devices. With this structure, a student from campus A will be able to use the network resources even if he is on campus B.

The difference between the Free Roaming system and this model are:

1. Authentication only versus Authentication and data transfer.

In this model, the only task is to authenticate users and authorize them to use the network. In the Free Roaming system, authentication is the first step. After this step, the users need to use this system and access their data without knowing where the data is located.

2. Web-based authentication versus Linux system authentication;

In this model, the authentication is through an HTTPS web page. In the Free Roaming system, users are authenticated when they login to the system.

3. Intranet versus Internet

In this model, all users belong to one network. In the Free Roaming system, users roam to all over the world.

Distributed File Systems

Network File System

NFS (Network File System) was designed by Sun Microsystems[6][12][13]. NFS allows its user to use a volume from a server as a sub-tree of the local Unix file system. NFS reaches this goal by using RPC (Remote Process Call) and VFS (Virtual File System). VFS is a set of generic consistent interfaces that enables the operating system to access many different types of file systems.

When an NFS client needs to access a file in the NFS server, the system calls will be translated to RPC procedures that use External Data Representation (XDR) to pass arguments and data between client and servers.

NFS file system is mounted on the local file system as a real file system so that the user can not distinguish the difference between accessing local file system and accessing NFS file system.

Andrew File System

AFS (Andrew File System) was designed in CMU (Carnegie Mellon University) in 1983[14]. AFS is similar to NFS based on its design purposes and uses RPC as the main mechanism for data transfer. AFS is based on an information-sharing backbone named "Vice". Vice acts as a single unit to the workstation, but it usually consists of several file servers

and local area networks. On each workstation, a process named "Venus" is responsible for coordinating file sharing. Venus's functions include finding files for the workstation, caching file locally, and act as an agent to operate on files. Both Vice and Venus are invisible to users. Workstations see them as a sub-tree of the local Unix file system. There are three versions of AFS, AFS-1, AFS-2 and AFS-3. AFS is an open source project now.

Coda File System

Coda is a descendent of AFS [15]. While AFS is a robust distributed file system, it cannot handle the inevitable problem of network failure. Network failure happens to wired network, and frequently happens to wireless network, such as blue-tooth, IEEE802.11 and RF networks. When the network failure happens, the client will not be able to access the data on the server. Some developers of AFS want to solve this problem and keep the excellent properties of AFS. Coda is the result of their effort. Coda successfully solved the network failure problem by automatically caching files onto local machines and writing them back onto the server when the network becomes available. This system is ideal for wireless networks.

Coda is a file system that deserves more detailed investigation for our Ubiquitous Computing system.

Samba File System

Samba [16] is used to connect Windows and Unix systems in the same network. It allows Windows and Unix systems to share resources with each other. It is very useful to a Local Area Network that has Windows and Unix systems together.

INTERMEZZO File System

InterMezzo file System[17] is a file system inspired by Coda file system. But it is not based on Coda file system code tree. It is an open source project and it has completely different code base.

InterMezzo File system focuses on providing high availability, for such environment like mobile computing system. For example, a laptop can work on files and disconnects from the network, and reconnect. A tool named "InterSync" for InterMezzo can re-synchronize files for the user. InterMezzo file system keeps a Kernel Modification Log(KML) as it modifies the file system. The KML collects the modification without scanning the file differences. InterSync periodically polls the server for changes and reintegrated these changes into client file system. InterSync uses HTTP as the protocol to get the KML from the server. Its work model is shown in Figure 6. Also because this is an open source file system, this file system

deserves more research and can be adapted into Free Roaming system.

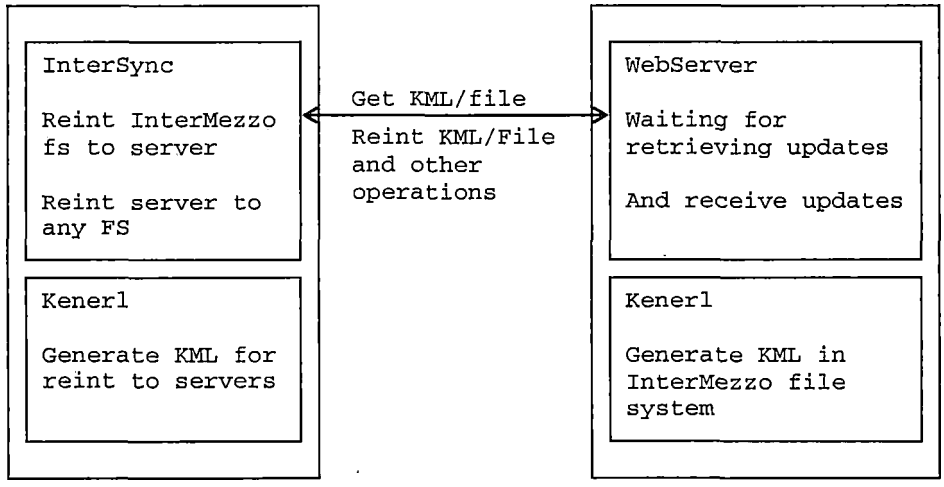

Figure 6. Inter-mezzo File System

Conclusion

For the proposed distributed authentication system discussed above, NIS cannot fulfill its requirements because it has the master-slave structure, and it requires compiling user data every time a new user is added. Active Directory is a part of the Windows server and cannot be adapted to the system. Kerberos, LDAP and Radius are closer to our requirements. Kerberos needs a more complex server and LDAP needs a well-structured data representation. They are not

selected to be used in this prototype. But they deserve further research, particularly LDAP. Radius is the selected system for its flexibility and extensibility.

Among the distributed file systems, NFS and Samba are designed to work in a LAN area. Since the Ubiquitous Computing system is intended to be deployed over a country or even over the world, these file systems will not be able to fulfill our requirements. AFS and Coda are closer to our requirements. They deserve more research to be adapted to the Free Roaming system.

Although AFS or Coda could be used in the system, some modifications are needed to move data with user. For the purpose of simplicity and ease of control, a file caching system based on FTP protocol is designed and implemented in the Free Roaming System.

InterMezzo file system deserves more research. It's a file system that is very close to our requirements. Its ability to synchronize files without scanning status of files is very desirable. How to adapt it to our system needs more deep research on its structure and source code.

The following chapters describe the Authentication System and the File Caching system used in the prototype for the Free Roaming System.

CHAPTER THREE

FREE ROAMING SYSTEM ARCHITECTURE

System View

A architectural diagram with more detail is shown in Figure 7. In this diagram, Radius servers are used to perform authentication. Like S.Keski-Kasari et al [10], proxy servers are used to construct a hierarchical and multi-master structure. SQL servers are used as a database to store user information. Data servers transfer user data on demand according to the result of authentication.

Network Elements

1. Data Servers

This project added a very important feature to the data server, which is the "data caching on demand agent", named "UFTP Agent". With this UFTP Agent, when a data server finds that a user is new, it will connect to his home data server, creates a local home directory for him and cache his data locally. With the UFTP Agent, the system allows users to travel around and access their personal data from anywhere.

Besides the UFTP Agent, the data server functions similarly to data server in the Umbilical Cord System. It

hosts users' data and provides a NFS service for the client to bind users' home directory.

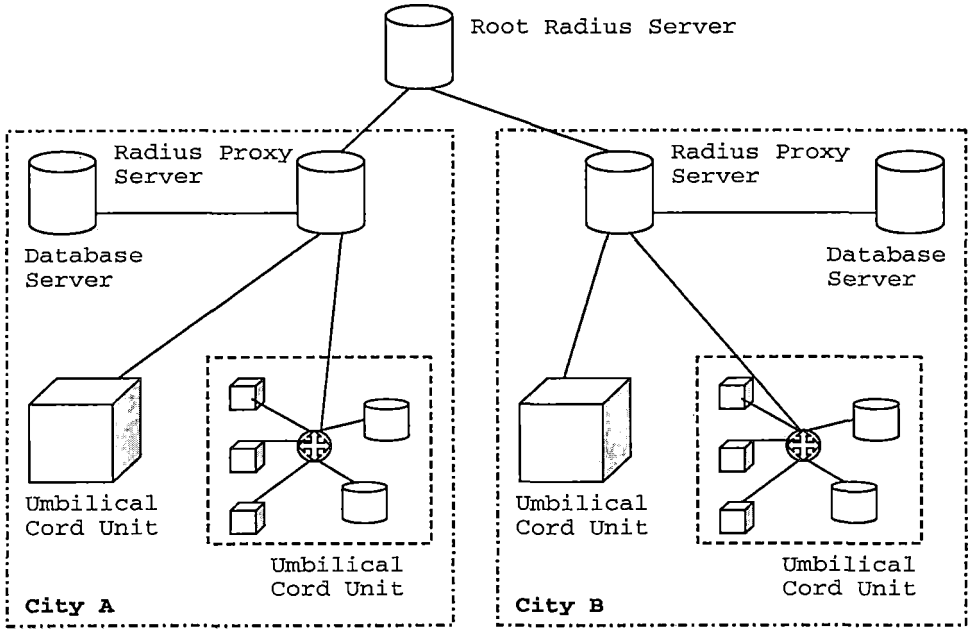

Figure 7. Detailed System Architecture

2. Operating System servers

Here, operating System servers have the same function as they did in the Umbilical Cord system. They provide configured operating system for clients to download.

3. Proxy Radius Servers

Proxy Radius Servers are new. They receive users' authentication request from a client, check the database and search for users' information. If they have the information,

they reply to the client. Otherwise, they forward the requests to the root Radius servers.

4. Root Radius Servers

Root Radius Servers have information of other proxy Radius servers. They process authentication requests from proxy Radius servers. After they receive a request from a proxy Radius server or workstation, they search the database and try to match the username-password pair.

A network element called "UFTP Messenger" is added to Proxy Radius Servers and Root Radius Servers. The main purpose of UFTP Messenger is to send a message to the UFTP Agent on the user's current data server after the user passes the authentication. The UFTP Agent connects to the user's home data server and downloads user's data. Parts of user data, including user's profile files, must be ready before he can use the computer.

5. SQL Servers

SQL servers store user information such as user name, password, home data server, current data server and last data server a user visited etc. They accept queries from Radius servers.

Roaming Example

An example of roaming between two different cities is illustrated in Figure 8.

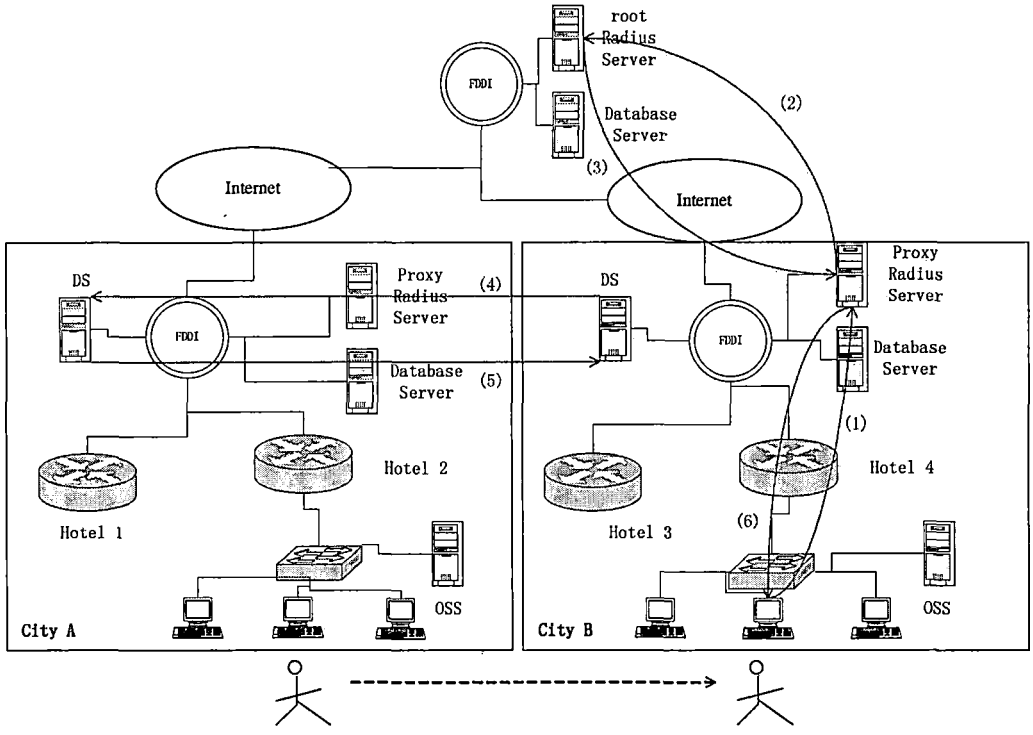

Figure 8. Roaming Example

In Figure 8, the two big squares represent two different cities. Workstations are deployed in hotels. Each hotel has a switch that connects all workstations together. There are routers that connect hotels to the city network.

In this example, assume a user from Los Angeles travels to New York and tries to login from a workstation of the Ubiquitous Computing system. After the user turns on the power button of the workstation, the workstation boots, gets the IP address from the local DHCP server, downloads the Operating System, and displays the login interface to the

user. The user inputs his user name and password, and starts the following roaming authentication process:

1. The PAM (Pluggable Authentication Module) module for Radius, which will be discussed later, sends the authentication packet out to the proxy Radius server.

2. The proxy Radius server receives the request and checks its database to see whether this user is local. If it finds the user's information, (2a) it checks the password, sends the "accept" reply to the workstation and a UFTP message to the current data server. Otherwise, (2b) it sends the authentication request to the next Radius server at a higher level. In this roaming example, we assume that this request goes to the root Radius server.

3. The root Radius server gets the request and checks the user information in its database. If the password is not correct, it sends a login-fail message, and the user fails on login. If the password is correct, the root Radius server sends the reply back to the proxy Radius server. It also sends a UFTP message to current data server.

4. The proxy Radius server gets the reply, interprets it and saves the user's information into its database. It then sends the authentication-success message to the client.

5. The local data server checks its directories and finds that it does not have this user's home directory. It

receives the new user message from the proxy Radius server, then creates a home directory for this user and caches the user's data locally. Future work includes caching most recently used data upon login, and then other data on demand.

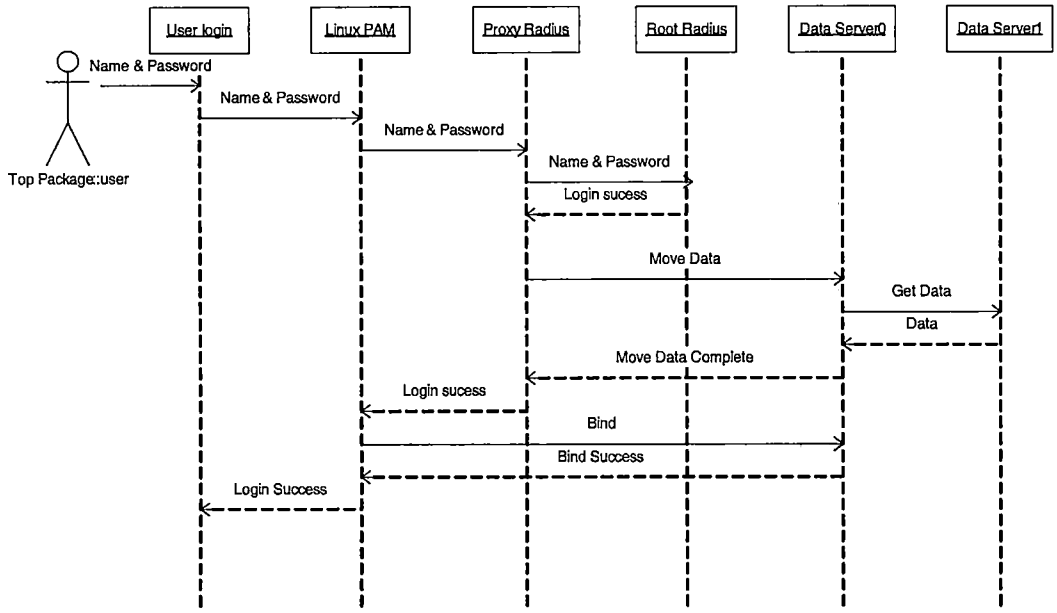

Figure 9. Login Sequence Diagram

6. The client gets the message and let the user login. It then tries to mount the user's home directory from the local data server.

7. The user can start working on his data. After he logs out from the client, the data will be saved back to his home server.

The sequence diagram in Figure 9 illustrates the above login sequence. In this sequence diagram, each step from starting from login that is described above is illustrated. All components, and message transferred between them are also illustrated.

CHAPTER FOUR

DISTRIBUTED AUTHENTICATION SYSTEM

System View

This chapter focuses on the Radius-based distributed authentication system that supports users to roam within the Free Roaming System.

FreeRadius is chosen as the authentication server in this distributed authentication system.

Linux-PAM (Pluggable Authentication Module) is a necessary component for using Radius to authenticate Linux requests [16]. Linux-PAM is a suite of shared libraries for Linux that enables the administrator to choose how applications authenticate users. Its working model is illustrated in figure 10. With Linux-PAM, various authentication methods can be chosen, including NIS, Radius, or password only. Linux-PAM also allows applications to have more than one method to authenticate users, so that when one method is not available, other users using other methods can still login to the system. For example, if the Radius servers are disconnected, the administrator can still login using local authentication password file.

MySQL database server is chosen to be the database system to store user information for this distributed authentication system. It stores user information such as user name, password, etc. Radius server queries this database when authenticating user login.

As stated before, the login process starts when a user input the user name and password. Please refer to Figure 9 for more details.

Pluggable Authentication Module and Linux

Most Linux distributions come with Linux-PAM installed. Specifically the client's base, RedHat Linux 9 comes with the Linux-PAM module. Linux-PAM can be configured to use either the single system file "/etc/pam.conf", or the "/etc/pam.d/" directory. In Redhat 9, directory-based configuration is used. "/etc/pam.d/system-auth" contains the configuration for the system login service.

Our goal here is to change the login to use radius authentication. Therefore, "/etc/pam.d/system-auth" needs to be changed to use radius module. The new configuration file is shown in appendix A.

Customized Radius Module

There is a PAM module ready to be used for Radius authentication. This module is provided in the form of

source code in C language and needs to be compiled before using it.

Original source code had some issues though. First, it failed to parse the server configuration file and hanged. It needed to be fixed.

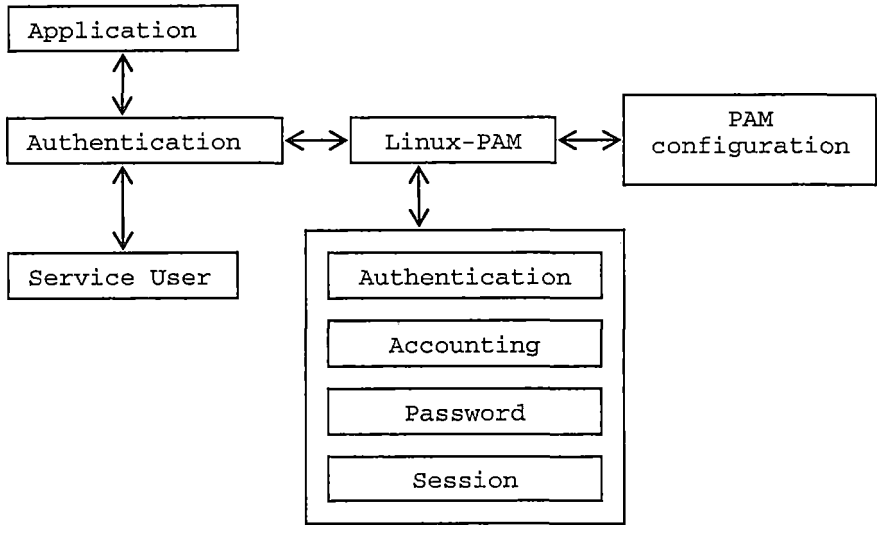

Figure 10. Pluggable Authentication Module

Second, and more important, the user name and password need of the user needs to be added to the user name and password database of the client. That means file "/etc/passwd" and "/etc/group" of the client need to be changed if this is a new user. Without this, the client will not let the user login and bind his home data folder

through NFS even the user has been successfully authenticated by the radius server.

After this module is compiled, the static library module "pam_radius_auth.so" needed to be copied into "/lib/security/" on the client. Then the client needed to be configured to use Radius as the authentication method for login. More details about configuring the client are described in [16]. As mention before, PAM configuration for system authentication in file "/etc/pam.d/system-auth" needs to be changed as in appendix A.

FreeRadius and MySQL

Radius can be configured to query user information from database servers during authentication and accounting process. The database also stores information including IP address of user's home data server, which will be sent to user's current data server when the authentication request is accepted.

Specifically, in our implementation, FreeRadius is configured to query MySQL database server. S. Bartlett [17] and J. Hassell [18], describe how to configure a FreeRadius server to query a MySQL database.

Tables and table structures in the database are attached as appendix E.

Configuring Proxy Radius Server

FreeRadius can act as a proxy server that forwards the requests to another authentication host. In this case, the proxy server acts as a client against another server. The configuration is stored in "/etc/raddb/proxy.conf". The "proxy.conf" file is attached in appendix D.

Changes to FreeRadius Server

To make FreeRadius server work as described previously, some changes have to be made. In this system, when a user is authenticated, FreeRadius server notifies the user's current data server, to let it know the IP address of the user's home data server. With the notification from the Radius server, the UFTP agent on the data server can go ahead and cache the user data locally. The caching system will be discussed further discussion in next chapter. A module that can send out this message needs to be added the FreeRadius server.

A TCP-based UFTP messenger is developed and added to FreeRadius server. It queries the database and searches for the user's home data server, and it sends a message to the user's current data server.

On the other side, in the data server, UFTP agent waits and listens for the incoming message from UFTP messenger.

After it verifies the message, it adds the user to its user database, and retrieves the data for the user from his home data server.

Packet Format of Messages from Messenger

UFTP Agent has a module which receives and processes messages from the UFTP messenger, which is a module added to the FreeRadius server. The message sent from UFTP messenger to UFTP Agent has the following format:

a. user name: 128 bytes;

b. password: 128 bytes;

c. home data server: 64 bytes;

d. current data server: 64 bytes.

UFTP Messenger and UFTP Agent is TCP-based socket program written using C language. The source code for UFTP Agent is attached as appendix C. The source code for UFTP messenger is attached as appendix B.

CHAPTER FIVE

DATA CACHING SYSTEM

System Requirements

To support a roaming user, the Free Roaming system needs to cache the user data to the local data server that the user visits. Then the client can mount the user's home directory via NFS. We need to have a way to move user data with the user. While other distributed file systems could be used as the data caching system, FTP protocol is selected in this prototype to develop a data caching system for its simplicity. Benefits of using FTP protocol are:

1. Simplicity and ease of control

   Using the FTP protocol to implement the data caching system is substantially simpler than hacking an existing distributed file system. A single file or the whole directory can be easily cached with this method.

2. Firewall issue

   Ports for FTP protocol are usually open for most firewalls.

3. Security

   Although FTP protocol itself is not a secure protocol, it is very easy to apply the same techniques and

upgrades to SFTP protocol for data transferring, which is very safe.

System View

In this Data Caching system, each data server has a data caching server named "UFTP server", and a data caching client named "UFTP client" running on it. When a roaming user logs into a client, the radius server sends a message to the local data server. This message includes IP address of current user's home data server. The local data server in turn launches his UFTP client to retrieve his data from his home data server. The topology is shown in Figure 5.

In this prototype, a regular FTP server is used as the UFTP server. On the client side, to fulfill the requirements of the prototype, a special client named "UFTP Agent" is implemented.

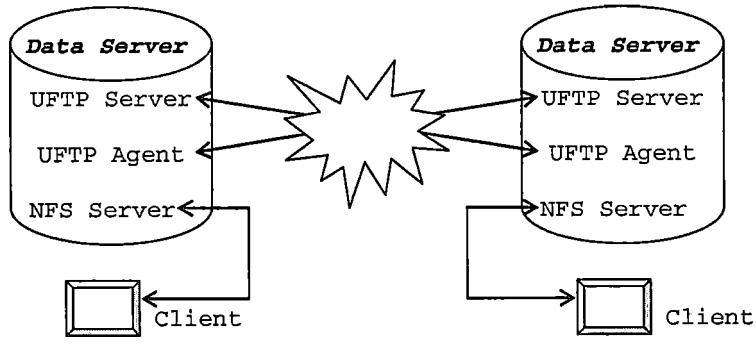

Figure 11. Data Caching System

From the view of the software architecture, the Ubiquitous File Caching System is designed to be a two-level system: FTP level functions and UFTP (Ubiquitous File Transfer Protocol) level functions. With this two level structure, even if the FTP protocol is replaced by another protocol, for example SFTP, the higher level interface would not change. FTP level functions implement the UFTP protocol. UFTP level functions are designed to use FTP level functions, and to provide an API for the data caching system.

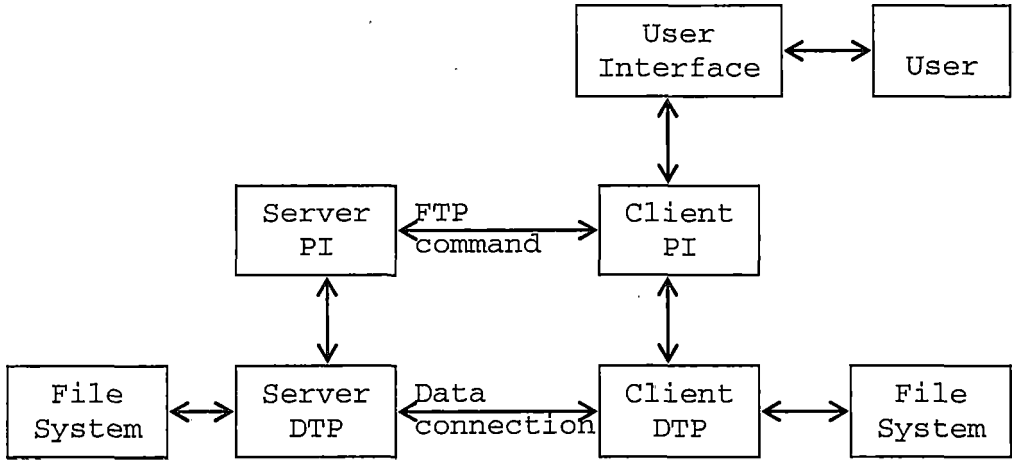

Figure 12. Work Model of File Transfer Protocol

File Transfer Protocol

FTP protocol, defined by RFC 959[3], is a TCP-based protocol for file transfer. Its work model is shown in Figure 11.

FTP protocol is an unusual protocol in that it uses two TCP ports, the data port is used for transferring data; the command port is used for transferring commands. The command port is usually designated by configuration, and the data port is negotiated by the server and the client. When the client and server start negotiating the data port, the FTP process can enter either one of two states: "passive" or "active" mode. In the active mode, the client chooses the data port number which usually is greater than 1024. Then the server initiates the data transfer by connecting to this port. If the client has a firewall configured, the firewall may block this incoming connection. On the other hand, in the passive mode, the client sends out "PASV" to indicate that it wants to use passive mode. After the server receives the request, a random port number greater than 1024 is selected for data connection. The client then connects to the server using this port. The benefit of passive mode is that the connection will not be blocked by the client side firewall. But it could be blocked by server side firewall. Fortunately, the server can be configured to have a range of

ports opened and listen on these ports only. Therefore, when deploying this Data Caching system, firewalls on both the client side and the server side need to be configured accordingly.

In the Free Roaming System, data caching process is automatically initiated by a roaming user. The initiation starts as follows:

1. When the user has been authenticated, the Radius Server sends a message to the user's current data server. This message contains the IP address of user's home data server.
2. If the data server determines that the user's home directory is not in its directory "/home", it initiates the FTP client to retrieve the home directory for the user.

Server of File Transfer Protocol

An off-the-shelf FTP server is used for this data caching system so that it does not need to be developed from scratch. In this prototype, VsFTP (Very Secure FTP) server is used as the FTP server for the following reasons:

1. Security
2. Open source
3. Ease of configuration

File Transfer Protocol Level Calls

In the FTP level, the following functions are implemented:

1. ftpHookup

This function is used to initiate an FTP session.

2. ftpLogin

This function is used for login.

3. ftpDataConnInitPassivMode

This function is used for starting a passive mode.

4. ftpDataConnGet

This function is used for initiating data connection.

5. ftpCommandEnhanced

This function is used for sending more complicated commands.

6. ftpReplyGet

This function is used to get reply from the server.

7. ftpXfer

This function is used to initiate a data transfer.

Ubiquitous File Transfer Protocol Level Calls

1. uftp_init

This function initializes a caching connection.

2. uftp_retrieveTree

This function retrieves the directory tree for a user.

3. uftp_getFile

This function retrieves a file from a user's home directory.

4. uftp_saveFile

This function restores a file onto user's home data server.

5. uftp_close

This function closes a caching session.

Among these functions, retrieving a tree is a recursive task. When the process starts, a "list" command is issued to a user's home directory by the FTP protocol. All files and directories, including their modes, will be sent from the server to the client. Then the client parses the input and retrieves each item of the input. According to the mode of a file and/or directory, the operation is different. If an item is a file, the client retrieves the file from the server. If an item is a directory, the client needs to retrieve the whole directory. Thus the procedure gets into the next level of recursively retrieving files.

The flow chart of this recursive procedure is shown in Figure 13.

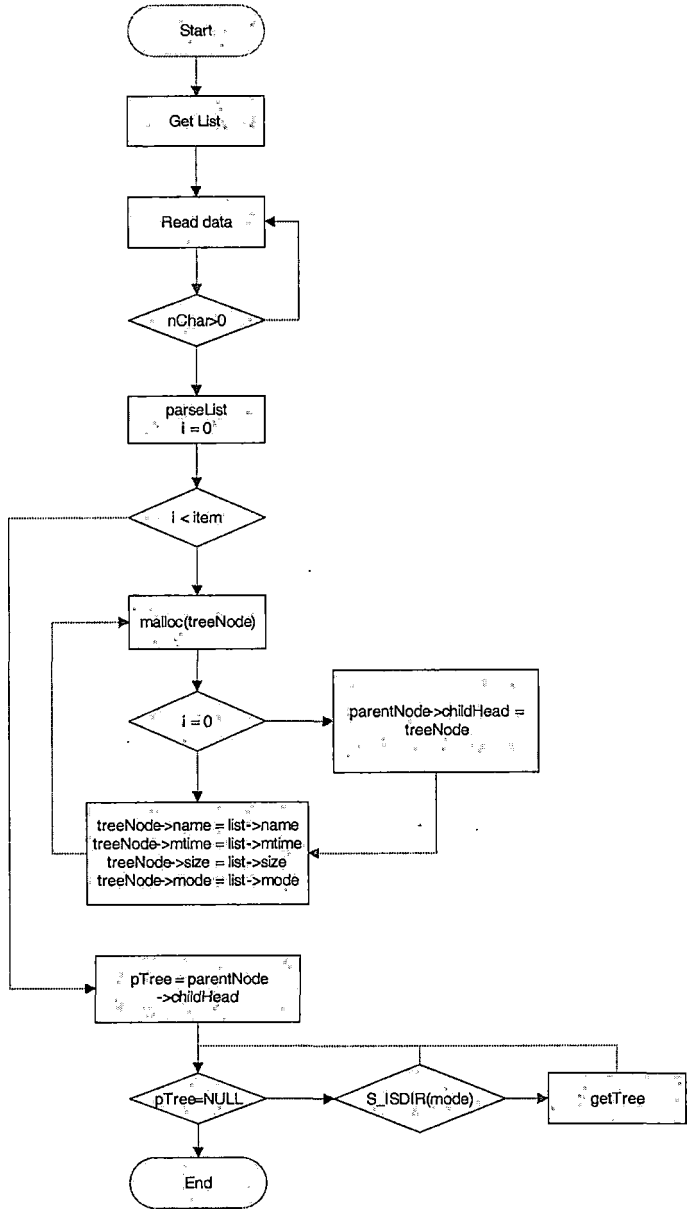

Figure 13. Block Diagram of GetTree

CHAPTER SIX

SUMMARY

Based on the Umbilical Cord system, a distributed authentication system and a data caching system are presented. The purpose is to setup a prototype of a Ubiquitous Computing system, which allows a user to roam within this computing system and provides a user with consistent user interface everywhere.

The distributed authentication system presented here is based on the RADIUS protocol. It supports multi-master and multi-level authentication. These features increase the extensibility and availability of the system. In the backend, a database is used to provide data for Radius server to query. How this system is implemented is described.

The distributed data caching system presented here is based on FTP protocol. It supports file caching between multiple data servers in the Internet. When the user roams to another place that is far away from his data server, his user data is cached to a data server close to him. So that he can access his user data everywhere without noticing this caching operation.

To combine the distributed authentication system and the distributed data caching system together, a UFTP messenger and a UFTP Agent are added. Some changes are made to the Radius server to fulfill the requirement. They all together work as a Free Roaming system.

CHAPTER SEVEN

FUTURE WORK

More Efficient Data Caching

Currently, the Data caching used is a directory-based caching system. When the user logs in from a remote data server, his whole home directory is cached locally. This method has two inefficiencies. First, caching a large directory leads to a long delay when the user logs in. Second, the user usually does not access all the files in his directory. A user usually opens one file at a time.

Whole-file transfer and page-transfer are two ways of data transfer with smaller granularity. Whole-file transfer copies the whole file to local data server once. User's operations including read and write happen locally, which is the same as the directory-based. Page-transfer reads and writes page by page of file to remote data server whenever the user's file system requires.

A whole-file-transfer caching is more appropriate for this system. When the user logs in at a remote site, all file names are cached locally, so that he can browse all files he has even using an X-based file browser. But when he open a file, the file can be cached locally on demand.

M. Satyanarayafian [21] has shown that block-based caching is not very efficient for a networked file system. One reason is that when the file is changed even one bit, the cache system has to save the entire file back. That increases network activity. Another reason is that the network can fail. Although the network could recover, the access to the file will be interrupted if page-based caching is used. This does not happen to file-based caching because it needs to access the file every for a while.

In the Ubiquitous System, user data is located on the Data Server. Caching happens between user's home data server and the local data server. The hop number between these two data servers could be very large. As a result, the possibility of network failure increases. Thus whole-file-transfer caching is more appropriate than block-based data caching.

More Secure Data Transfer

FTP is not a secure protocol. User name and password are transferred between client and server using unencrypted text, which makes it vulnerable to many kinds of attack.

Described by J. Galbraith et al [22], SFTP (Secure Shell FTP), as part of SSH (Secure Shell), is a better architecture for network login and file transfer. SSH architecture is defined by T. Ylonen et al [23]. It mainly

consists of transport layer protocol T. Ylonen et al[24], user authentication protocol T. Ylonen et al[25], and connection protocol T. Ylonen et al[26]. The work model of SSH is illustrated in Figure 14.

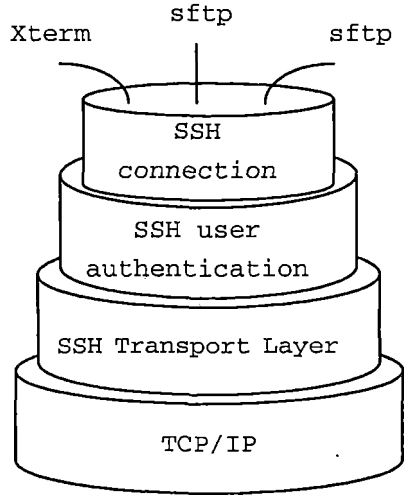

Figure 14. Work Model of Secure Shell

The purpose of SSH is to transfer data securely over unsecured protocols, for example TCP/IP protocol stack. Figure 14 shows that the SSH protocol sets up a secure tunnel over TCP/IP stack, for multi-purpose application.

More Available and Reliable Services

To improve the availability and reliability, multiple and redundant radius servers, database servers, data servers

and operating system servers are needed. More research and investigation in this area is needed.

InterMezzo File system

As stated in chapter one, InterMezzo file system is very interesting file system which is very close to the requirement of this project. Although this project has implemented a FTP-base file caching system, it's highly recommended that future researchers would pay some time to look deeply in how InterMezzo File System is implemented and see if it is possible to adapt it into this Free Roaming System.

User Data Backup

Architecturally, being a distributed system, this system could be more secure than a PC at home because user data can easily have a remote backup depending on the backup strategy. It's quite natural to design the system to have more than one backup for a user's home directory. This remote backup can be very useful in case of disaster recovery. More research is needed on this topic.

CHAPTER EIGHT

CONCLUSION

Based on the Umbilical Cord system, a Free Roaming system is brought out and implemented. This Free Roaming consists of a distributed authentication system, a data caching system, and a message system between them. It allows user to roam within this system and access his data everywhere. Together with the Umbilical Cord system, a Ubiquitous Computing system is functionally completed as a prototype, and is ready to be deployed into the Internet.

The distributed authentication system is based on the RADIUS protocol[6]. It supports multi-master and multi-level authentication. These features increase the extensibility and availability of the system.

The distributed data caching system presented here is based on FTP protocol[3]. But it can be change to use any protocol. It supports file caching between multiple data servers in the Internet.

At the end, some suggestions to improve the system are given.

APPENDIX A

SYSTEM-AUTH FILE

```
#filename : system-auth
#%PAM-1.0
auth       required   pam_securetty.so
auth       required   pam_radius_auth.so
auth       required   pam_stack.so service=system-auth
auth       required   pam_nologin.so
account    required   pam_stack.so service=system-auth
password   required   pam_stack.so service=system-auth
session    required   pam_stack.so service=system-auth
session    optional   pam_console.so
```

APPENDIX B

UFTP MESSENGER

```
/*
** messenger.c -- a client that sends message to an agent on DS *server.
The message includes, the a ubiquitous client's password,
*username,current Data server address, home data server address. So *the
DS server will go ahead to get the user data for the currently *log in
user.
*/


#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>


#define AGENT_PORT 3490 // the port client will be connecting to


#define MAXDATASIZE 2048 // max number of bytes we can get at once


#define USERNAME_SIZE 128
#define PASSWORD_SIZE 128
#define IP_SIZE 64


typedef struct message_S{
  char * hostname;
  unsigned char username[128];
  unsigned char password[128];
  unsigned char homeDS[64];
  unsigned char currDS[64];
} messsage;


int send_message(message * msg)
{
   int sockfd, numbytes;
   char buf[MAXDATASIZE];
   struct hostent *he;
```

```
struct sockaddr_in their_addr; // connector's address information
unsigned char * pointer;

if ((he=gethostbyname(msg->hostname)) == NULL) {//get the host
   herror("gethostbyname");
   exit(1);
}
/* prepare the connection */
if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
   perror("socket");
   exit(1);
}
their_addr.sin_family = AF_INET;    // host byte order
their_addr.sin_port = htons(AGENT_PORT); //short,network order
their_addr.sin_addr = *((struct in_addr *)he->h_addr);
memset(&(their_addr.sin_zero), '\0', 8);  // zero the rest

if (connect(sockfd, (struct sockaddr *)&their_addr,
                            sizeof(struct sockaddr)) == -1) {
   perror("connect");
   exit(1);
}

/* compose the buffer */
pointer = buf;
memcpy(pointer, msg->username, USERNAME_SIZE);
pointer += USERNAME_SIZE;
memcpy(pointer , msg->password, PASSWORD_SIZE);
pointer += PASSWORD_SIZE;
memcpy(pointer , msg->homeDS, IP_SIZE);
pointer += IP_SIZE;
memcpy(pointer, msg->currDS, IP_SIZE);
pointer += IP_SIZE;

/* send the packet out */
if ((numbytes=send(sockfd, buf, MAXDATASIZE-1, 0)) == -1) {
   perror("send");
   exit(1);
```

```
    }
    printf("Message Send to : %s", msg->currDS);
    close(sockfd);
    return 0;
}
```

APPENDIX C

UFTP AGENT

```
/*
** uftpAgent.c -- This file contains the main file for uftpAgent.
* uftpAgent is an agent that resides on a data server. It's a deamon
* receives messages from uftp messengers and retrieves home data for a
* user that login to a client of Ubiquitous
* Computing system close to it, which will use NFS bind to bind the home
directory.
*
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>
#include "ftp.h"
#include "string.h"


#define AGENT_PORT 3490   // the port users will be connecting to
#define MAXDATASIZE 2048  // max number of bytes we can get at once

#define USERNAME_SIZE 128
#define PASSWORD_SIZE 128
#define HOSTNAME_SIZE 128
#define IP_SIZE 64

typedef struct message_S{
  char * hostname;
  unsigned char username[128];
  unsigned char password[128];
  unsigned char homeDS[64];
```

```
 unsigned char currDS[64];
} message;

char current_user[USERNAME_SIZE];
char current_password[PASSWORD_SIZE];
char current_host[HOSTNAME_SIZE];


#define BACKLOG 10     // how many pending connections queue will hold

void sigchld_handler(int s)
{
   while(waitpid(-1, NULL, WNOHANG) > 0);
}

int main(void)
{
   int sockfd, new_fd;  // listen on sock_fd, new connection on new_fd
   struct sockaddr_in my_addr;    // my address information
   struct sockaddr_in their_addr; // connector's address information
   socklen_t sin_size;
   struct sigaction sa;
   int yes=1;
   unsigned char buffer[MAXDATASIZE];
   message *newmsg;


   if ((sockfd = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
      perror("socket");
      exit(1);
   }

    if   (setsockopt(sockfd,SOL_SOCKET,SO_REUSEADDR,&yes,sizeof(int))==-1)
    {
      perror("setsockopt");
      exit(1);
   }
```

```
   my_addr.sin_family = AF_INET;         // host byte order
   my_addr.sin_port = htons(AGENT_PORT); //short, network byte order
   my_addr.sin_addr.s_addr= ANADDR_ANY; //automatically fill with my IP
   memset(&(my_addr.sin_zero), '\0', 8); // zero the rest of the struct

   if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct
sockaddr))== -1) {
      perror("bind");
      exit(1);
   }

   if (listen(sockfd, BACKLOG) == -1) {
      perror("listen");
      exit(1);
   }

   sa.sa_handler = sigchld_handler; // reap all dead processes
   sigemptyset(&sa.sa_mask);
   sa.sa_flags = SA_RESTART;
   if (sigaction(SIGCHLD, &sa, NULL) == -1) {
      perror("sigaction");
      exit(1);
   }
   printf("UFTP Agent ready...\n");
   while(1) {  // main accept() loop
      sin_size = sizeof(struct sockaddr_in);
      if((new_fd=accept(sockfd,(structsockaddr  *)&their_addr,&sin_size))
== -1) {
         perror("accept");
         continue;
      }
      printf("server: got connection from %s\n",
                                 inet_ntoa(their_addr.sin_addr));
      if (!fork()) { // this is the child process
         close(sockfd); // child doesn't need the listener
         memset(buffer, 0, MAXDATASIZE);
         if (recv(new_fd, buffer, MAXDATASIZE, 0) == -1){
            perror("RECV");
         }
```

```
            else {
              /* Parse the data we receive */
               newmsg = (message *)malloc(sizeof(message));
               memset(newmsg, 0, sizeof(message));

               parseMessage(buffer, newmsg);
               retrieveData(newmsg);
               free(newmsg);
            }
            close(new_fd);
            exit(0);
         }

         close(new_fd);  // parent doesn't need this
      }
      return 0;
}

/*
** Parse the message we get from messenger,
 * input
 */
void parseMessage(unsigned char * buf, message * msg){
   unsigned char * pointer;
   pointer = buf;
   memcpy(msg->username, pointer, USERNAME_SIZE);
   pointer = pointer + USERNAME_SIZE;
   memcpy(msg->password, pointer, PASSWORD_SIZE);
   pointer = pointer + PASSWORD_SIZE;
   memcpy(msg->homeDS, pointer, IP_SIZE);
   memcpy(msg->currDS, pointer, IP_SIZE);
   return;
}

/*
 * Retrieve data for a user based on the message we get from the radius
server
*/
```

```
int retrieveData(message *msg){
   char *host;        /* name of server host */
   char *user;        /* user name for host login */
   char *passwd;      /* password for host login */
   char *acct;        /* account for host login */
   char *cmd;         /* command to send to host */
   char *dirname;     /* directory to 'cd' to before sending command */
   char *filename;    /* filename to send with command */
   int CtrlSock;      /* control socket fd */
   int DataSock;      /* data socket fd */
    STATUS status;

//   host = inet_ntoa(msg->homeDS);
   host = msg->homeDS;
   strcpy(current_host, host);
   strcpy(current_user, msg->username);
   strcpy(current_password, msg->password);
   printf("retrieveData:Retrieving data from %s, for %s, password %s\n",
                   current_host, current_user, current_password);
   uftp_init(current_host,current_user,current_password,0);
   uftp_retrieveTree(current_user);
   uftp_close();
   return 0;
}
```

APPENDIX D

RADIUS PROXY SERVER CONFIGURATION

1.configure local realms whose authentication requests are not forward:

```
realm losangels{
type = radius
authhost = LOCAL
accthost = LOCAL
secret = uceverywhere
}
```

2.configure remote realms whose authentication requests needs to be forwarded, add the following section to proxy.conf:

```
realm newyork{
type = radius
authhost = radius.uceverywhere.com
accthost = radius.uceverywhere.com
secret = uceverywhere
}
```

APPENDIX E

DATABASE STRUCTURE

Currently, the MySql database has the following tables:

1. homeserver;
2. nas;
3. radacct;
4. radcheck;
5. radgroupcheck;
6. radgroupreply;
7. radpostauth;
8. radreply;
9. usergroup;

There fields are listed below.

Table 1. homeserver

Field	Type	Null	Key	Default	Extra
UserName	varchar(64)	YES		NULL	
HomeServer	varchar(64)	YES		NULL	

Table 2. nas

Field	Type	Null	Key	Default	Extra
Id	int(10)		PRI	NULL	auto_increment
Nasname	varchar(128)		MUL		
Shortname	varchar(32)	YES		NULL	
Type	varchar(30)	YES		NULL	
Ports	int(5)	YES		NULL	
Secret	varchar(60)	YES		Secret	
Community	varchar(50)	YES		NULL	
Descriptio n	varchar(200)	YES		RADIUS Client	

Table 3. radacct

Field	Type	Null	Key	Default	Extra
RadAcctId	bigint(21)		PRI	NULL	auto_incre ment
AcctSessionID	varchar(32)		MUL		
AcctUniqueID	varchar(32)		MUL	NULL	
UserName	varchar(64)		MUL	NULL	
Realm	varchar(64)	YES			

Field	Type	Null	Key	Default	Extra
NASIPAddress	varchar(15)		MUL		
NASPortId	int(12)	YES		NULL	
NASPortType	varchar(32)	YES		NULL	
AcctStartTime	datetime			MUL	0000-00-00 00:00:00
AcctSTopTime	datetime			MUL	0000-00-00 00:00:00
AcctSessionTime	int(12)	YES		NULL	
ConnectInfoStart	varchar(32)	YES		NULL	
ConnectInfoStop	varchar(32)	YES		NULL	
AcctInputOctets	Bigint(12)	YES		NULL	
AcctOnputOctets	Bigint(12)	YES		NULL	
CalledStationID	varchar(50)				
CallingStationID	varchar(50)				
AcctTerminateCause	varchar(32)				
ServiceType	varchar(32)	YES		NULL	
FramedProtocol	varchar(32)	YES		NULL	
FramedIPAddress	varchar(15)		MUL		
AcctStartDelay	int(12)	YES		NULL	
AcctStopDelay	int(12)	YES		NULL	

Table 4. radcheck

Field	Type	Null	Key	Default	Extra
Id	int(11) unsigned		PRI	NULL	auto_increment
UserName	varchar(64)		MUL		
Attribute	varchar(32)	YES		NULL	
Op	varchar(2)			=	
Value	varchar(253)	YES		NULL	

Table 5. radgroupcheck

Field	Type	Null	Key	Default	Extra
Id	int(11) unsigned		PRI	NULL	auto_increment
GroupName	varchar(64)		MUL		
Attribute	varchar(32)	YES		NULL	
Op	varchar(2)			=	
Value	varchar(253)	YES		NULL	

Table 6. radgroupreply

Field	Type	Null	Key	Default	Extra
Id	int(11) unsigned		PRI	NULL	auto_increment
GroupName	varchar(64)		MUL		
Attribute	varchar(32)	YES		NULL	
Op	varchar(2)			=	
Value	varchar(253)	YES		NULL	
prio	int(10)unsigned	YES		0	

Table 7. radpostauth

Field	Type	Null	Key	Default	Extra
Id	int(11)		PRI	NULL	auto_increment
User	varchar(64)				
Pass	varchar(32)				
Reply	varchar(2)				
date	timestamp	YES		CURRENT_TIMESTAMP	

Table 8. radreply

Field	Type	Null	Key	Default	Extra
Id	int(11) unsigned		PRI	NULL	auto_increment
UserName	varchar(64)		MUL		
Attribute	varchar(32)				
Op	varchar(2)			=	
Value	varchar(253)				

Table 9. usergroup

Field	Type	Null	Key	Default	Extra
Id	int(11) unsigned		PRI	NULL	Auto_increment
UserName	varchar(64)		MUL		
GroupName	varchar(64)				

REFERENCES

[1] J. E. Warshawsky, "Umbilical Cord, A System for Ubiquitous Computing", CSUSB MSCS Graduate Project, June 2004.

[2] C. Rigney, A. Rubens, W. Simpson, S. Willens, "RFC 2138 -Remote Authentication Dial In User Service (RADIUS)", April 1997.

[3] J.Postel, J. Reynolds, "RFC 959 - File Transfer Protocol", October 1985.

[4] J. G. Steiner, B. C. Neuman, and J. I. Schiller. "Kerberos: An Authentication Service for Open Network Systems". In Proceedings of the Winter 1988 Usenix Conference. February 1988.

[5] W. Yeong, T. Howes and S. Kille, "RFC 1777 - Lightweight Directory Access Protocol", Mar, 1995.

[6] Microsoft, "Active Directory Architecture", Microsoft, <http://www.microsoft.com/technet/prodtechnol/windows2000serv/technologies/activedirectory/deploy/projplan/adarch.mspx>.

[7] H. Stern, "Managing NFS and NIS",O'Reilly & Associates, INC. June 1991.

[8] C. Rigney, S. Willens, A. Rubens, W. Simpson, "RFC 2865-Remote Authentication Dial In User Service (RADIUS)", June 2000.

[9] M. Gallapher, "Mobile Telecommunications Networking

with IS-41", 1997, McGraw-Hill Book Company.

[10] Y. Lin, "Mobility Management for Cellular Telephony Networks", IEEE Parallel & Distributed Technology: Systems & Technology, Volume 4, Issue 4, Pages: 65-73, December 1996.

[11] S. Keski-Kasari, K. Huhtanen and J. Harju, "Applying Radius-based Public Access Roaming in the Finnish University Network (FUNET)", TERENA Networking conference 2003, May 2003.

[12] B. Callaghan, B. Pawlowski, P. Staubach, "RFC1813-NFS Version 3 Protocol Specification", June 1995.

[13] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C.Beame, M. Eisler, D. Noveck, "RFC3530-Network File System (NFS) version 4 Protocol", April 2003.

[14] M. Satyanarayanan, "Scalable, Secure, and Highly Available Distributed File Access", IEEE Computer, May 1990, Vol.23, No.5.

[15] M. Satyanarayanan, "The Evolution of Coda", ACM Transactions on Computer Systems, Vol. 20, No. 2, May 2002, Pages 85-124.

[16] J. Ts, R. Eckstein, and D. Collier-Brown, "Using Samba, 2nd Edition", O'Reilly & Associates, February 2003.

[17] P.J. Braam, "InterMezzo: File Synchronization with InterSync", March, 2002, <http://www.inter-mezzo.org/docs/intersync.pdf>

[18] A. G. Morgan, "The Linux-PAM System Administrators' Guide", June 2002, <http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam.html>.

[19] S. Bartlett, "FreeRadius and MySQL howto notes", March 2005, <http://www.frontios.com/freeradius.html>

[20] J. Hassell, "RADIUS", O'Reilly Association Inc., October 2002.

[21] M. Satyanarayafian, "On the Influence of Scale in a Distributed System", October 1998.

[22] J. Galbraith, T. Ylonen and S. Lehtinen, "SSH File Transfer Protocol", draft-ietf-secsh-filexfer-04.txt, October 2001.

[23] T. Ylonen and C. Lonvick, "SSH Protocol Architecture", draft-ietf-secsh-architecture-12.txt, August 2002.

[24] T. Ylonen and C. Lonvick, "SSH Transport Layer Protocol", draft-ietf-secsh-transport-24.txt, March 2005.

[25] T. Ylonen and C. Lonvick, "SSH Authentication Protocol", draft-ietf-secsh-userauth-17.txt, March 2005.

[26] T. Ylonen and C. Lonvick, "SSH Connection Protocol", draft-ietf-secsh-connect-25.txt, March 2005.